AMENDMENTS TO THE CLAIMS

This listing of claims will replace all prior versions of claims in the application:

Listing of Claims:

1. (Previously presented) A system for mitigating problems associated with automatic execution of initialization code, the system comprising:

an initialization method activator that calls a class initialization method at a pre-determined execution point; and

a deadlock analyzer that determines whether running the class initialization method will produce a deadlock.

2. (Previously presented) The system of claim 1, where the initialization method activator checks whether a class is initialized and, if the class is not initialized, calls the class initialization method.

3. (Previously presented) The system of claim 2, where the deadlock analyzer determines whether calling the class initialization method will generate a deadlock.

4. (Original) The system of claim 3, where the pre-determined execution point is at least one of one of a caller's just in time compilation time, a callee's just in time compilation time, an initial field access time, an initial method access time, an initial static field access time and a first access of pre-compiled code where no just in time compilation occurs.

5. (Previously presented) The system of claim 1, where the initialization method detector associates initialization check code with one or more components associated with a runtime environment, where the initialization check code determines whether a class has been initialized.

6. (Previously presented) The system of claim 5, where the initialization check code determines whether calling the class initialization method will generate a deadlock, and if a deadlock will be generated, to resolve the deadlock.

7. (Original) The system of claim 6, where the initialization check code is run at one of a caller's just in time compilation time, a callee's just in time compilation time, a first field access time, a first method access time, a first static field access time and a first access of pre-compiled code where no just in time compilation occurs.

8. (Original) The system of claim 1, where the deadlock analyzer analyzes a wait for graph.

9. (Previously presented) The system of claim 8 where the deadlock analyzer resolves a deadlock associated with running the class initialization method.

10. (Original) The system of claim 9, where the deadlock analyzer adds and/or removes one or more nodes and/or arcs from the wait for graph.

11. (Previously presented) The system of claim 8, further comprising a semantic analyzer that analyzes a semantic type associated with the class initialization method, where the semantic analyzer provides information concerning a desired initialization check time to the initialization method activator.

12. (Original) The system of claim 11, where the semantic type is one of "exact" and "before field initialization".

13. (Previously presented) The system of claim 11, further comprising a domain uniqueness analyzer that analyzes the uniqueness of one or more domains with which the class initialization method and/or the class will interact, where the uniqueness analyzer provides information concerning a desired initialization check time to the initialization method activator.

14. (Original) The system of claim 13, where the domain uniqueness is one of "normal" and "domain neutral".

15. (Previously presented) A computer readable medium containing computer executable components of a system for mitigating problems associated with automatic execution of initialization code, the system comprising:

an initialization method activating component that calls the class initialization method at a pre-determined execution point; and

a deadlock analyzer that determines whether running the class initialization method will produce a deadlock.

16. (Previously presented) A computer readable medium containing computer executable components of a system for mitigating problems associated with automatic execution of initialization code, the system comprising:

a semantic analyzing component that determines a semantic type associated with the initialization method;

a domain uniqueness analyzing component that determines a uniqueness type associated with one or more application domains with which the class will interact;

a deadlock analyzing component that determines whether calling the initialization method will create a deadlock and resolves the deadlock; and

an initialization method activating component that calls the initialization method at a pre-determined execution point, where the pre-determined execution point depends on, at least in part, the semantic type and the domain uniqueness.

17. (Previously presented) A method for mitigating problems associated with automatic execution of class initialization code, the method comprising:

determining whether a class has an initializing method;

determining when the initializing method should be run;

associating initialization check code with one or more components associated with a runtime, the check code determines whether a class is initialized;

determining whether calling the initializing method will generate a deadlock and if calling the initializing method will generate a deadlock, resolving the deadlock; and

calling the class initializing method.

18. (Original) The method of claim 17, where determining when the initializing method should be run comprises:

analyzing semantic information associated with the initializing method.

19. (Original) The method of claim 18, where the semantic information comprises an identifier that identifies whether the initializing method desires "exact" or "before field initialization" behavior.

20. (Original) The method of claim 19, where determining when the initializing method should be run further comprises analyzing domain uniqueness information associated with one or more domains with which the class initialization code will interact.

21. (Original) The method of claim 20, where the domain uniqueness information comprises an identifier that identifies whether the initializing method is associated with a "normal" or a "domain neutral" environment.

22. (Original) The method of claim 17, where determining whether calling the initializing method will generate a deadlock comprises:

attempting to acquire an initialization lock associated with the class to be initialized, and if the initialization lock cannot be acquired, identifying a holding thread that is holding the initialization lock;

locating a node associated with the holding thread, where the node is located in a wait for graph; and

analyzing the wait for graph to determine whether a deadlock exists.

23. (Original) The method of claim 22, where analyzing the wait for graph to determine whether a deadlock exists comprises traversing the wait for graph starting at the node associated with the holding thread and determining whether a cycle is detected in the wait for graph.

24. (Original) The method of claim 23, where resolving the deadlock comprises:

acquiring a lock associated with the wait for graph;

if a detecting thread that identifies the deadlock previously added one or more arcs and/or nodes to the wait for graph, removing the one or more arcs and/or nodes from the wait for graph;

releasing the lock associated with the wait for graph; and

the detecting thread interacting with the class as though the class was initialized.

25. (Original) The method of claim 17, where determining whether calling the initializing method will generate a deadlock comprises:

an acquiring thread attempting to acquire an initialization lock associated with the class to be initialized, and, if the lock can not be acquired, determining whether there is a thread waiting for the acquiring thread to complete its processing and release the initialization lock.

26. (Original) The method of claim 25, where resolving the deadlock comprises:

if it was determined that there was a thread waiting for the acquiring thread to complete its processing and release the initialization lock, then returning and interacting with the partially initialized state of the class as though the class were initialized, otherwise, blocking until the class becomes initialized.

27. (Previously presented) A computer readable medium containing computer executable instructions that perform a method for mitigating problems associated with automatic execution of class initialization code, the method comprising:

determining whether a class has an initializing method;

determining when the initializing method should be run;

inserting initialization check code into one or more components associated with a runtime;

determining whether calling the initializing method will generate a deadlock and if calling the initializing method will generate a deadlock, resolving the deadlock; and

calling the class initializing method.

28. (Original) A method for mitigating problems associated with automatic execution of class initialization code, the method comprising:

determining whether a class has an initializing method;

determining when the initializing method should be run, where determining when the initializing method should be run comprises:

analyzing semantic information associated with the initializing method, where the semantic information comprises an identifier that identifies whether the initializing method desires "exact" or "before field initialization" behavior; and

analyzing domain uniqueness information associated with one or more domains with which the initializing method will interact, where the domain uniqueness information comprises an identifier that identifies whether the initializing method is operating in a "normal" or a "domain neutral" environment;

associating initialization check code with one or more components associated with a runtime;

determining whether calling the initializing method will generate a deadlock;

resolving the deadlock; and

calling the class initializing method.

29. (Original) The method of claim 28, where determining whether calling the initializing method will generate a deadlock comprises:

attempting to acquire an initialization lock associated with the class to be initialized, and if the initialization lock cannot be acquired, identifying a holding thread that is holding the initialization lock;

locating a node associated with the holding thread, where the node is located in a wait for graph; and

analyzing the wait for graph to determine whether a deadlock exists, where analyzing the wait for graph to determine whether a deadlock exists comprises:

traversing the wait for graph starting at the node associated with the holding thread and determining whether a cycle is detected in the wait for graph.

30. (Original) The method of claim 29, where resolving the deadlock comprises:

acquiring a lock associated with the wait for graph;

if the thread that identifies the deadlock previously added one or more arcs and/or nodes to the wait for graph, removing the one or more arcs and/or nodes from the wait for graph;

releasing the lock associated with the wait for graph; and

in the thread that detected that it could not initialize the class because a deadlock existed with another thread that was initializing the class, interacting with the class as though it was initialized.

31. (Original) A computer readable medium containing computer executable instructions for performing a method for mitigating problems associated with automatic execution of class initialization code, the method comprising:

determining whether a class has an initializing method;

determining when the initializing method should be run, where determining when the initializing method should be run comprises:

analyzing semantic information associated with the initializing method, where the semantic information comprises an identifier that identifies whether the initializing method desires "exact" or "before field initialization" behavior; and

analyzing domain uniqueness information associated with one or more environments with which the initializing method will interact, where the domain uniqueness information comprises an identifier that identifies whether the initializing method will interact with a "normal" or a "domain neutral" environment;

associating initialization check code with one or more components associated with a runtime;

determining whether calling the initializing method will generate a deadlock;

resolving the deadlock; and

calling the class initializing method.

32. (Previously presented) A system for mitigating problems associated with automatic execution of class initialization code, the method comprising:

means for identifying a constructor associated with a class;

means for scheduling the running of the constructor;

means for adding code into one or more components generated by a runtime, the code identifies whether a class is initialized;

means for detecting deadlocks between constructors;

means for resolving deadlocks between constructors; and

means for invoking a constructor.

33. (Original) A data packet adapted to be transmitted between two or more components, the data packet comprising:

information associated with one or more nodes associated with a wait for graph, where the nodes model one or more threads to be analyzed to determine whether class initialization code will generate a deadlock; and

information associated with one or more arcs associated with a wait for graph, where the arcs model one or more wait for relationships between one or more of the nodes.

34. (Previously presented) A data packet adapted to be transmitted between two or more components, the data packet comprising:

a first field that holds information concerning the identity of a thread that is attempting to initialize a class;

a second field that holds information concerning the identity of one or more threads that are waiting for a class to be initialized; and

a third field that holds information concerning the initialization status of a class to facilitate deadlock detection and resolution.